

Contents

1 Routine/Function Prologues	2
1.1 Fortran: Module Interface lisdrv_module.F90 (Source File: lisdrv_module.F90)	2
1.1.1 ld_domain_init (Source File: lisdrv_module.F90)	2
1.1.2 setup_timeMgr (Source File: lisdrv_module.F90)	3
1.1.3 LIS_ticktime (Source File: lisdrv_module.F90)	3
1.1.4 LIS_allocate_memory (Source File: lisdrv_module.F90)	4
1.1.5 setnch (Source File: lisdrv_module.F90)	6
1.1.6 getdomain (Source File: lisdrv_module.F90)	6
1.1.7 getlsm (Source File: lisdrv_module.F90)	7
1.1.8 getnch (Source File: lisdrv_module.F90)	7
1.1.9 getnc (Source File: lisdrv_module.F90)	8
1.1.10 getnr (Source File: lisdrv_module.F90)	8
1.1.11 getmaxt (Source File: lisdrv_module.F90)	8
1.1.12 getforcing (Source File: lisdrv_module.F90)	9
1.1.13 getnmif (Source File: lisdrv_module.F90)	9
1.1.14 LIS_endofrun (Source File: lisdrv_module.F90)	9
1.1.15 dist_gindex (Source File: lisdrv_module.F90)	10

1 Routine/Function Prologues

1.1 Fortran: Module Interface lisdrv_module.F90 (Source File: lisdrv_module.F90)

Main program for LIS This module contains interfaces and subroutines that control program execution.

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

USES:

```
use lis_module
use grid_module
use time_manager
use tile_spmdMod
use tile_module
```

INTERFACE:

```
interface LIS_domain_init
    module procedure ld_domain_init
end interface
```

1.1.1 ld_domain_init (Source File: lisdrv_module.F90)

Calls routines to read the card file, and initialize the time manager

INTERFACE:

```
subroutine ld_domain_init()
```

CONTENTS:

```
#if (defined SPMD)
    call spmd_init()
#endif
#if ( defined OPENDAP )
    call readcard()
    call define_domains()
    call read_domain(lis%d%domain)
#else
    if ( masterproc ) then
        call readcard()
        call define_domains()
```

```

        call read_domain(lis%d%domain)
    endif
#endif
!-----
! we have read in the time parameters. Use them to initialize ESMF
! time manager
!-----
if ( masterproc ) then
    call setup_timeMgr()
    lis%t%endtime = 0
endif

```

1.1.2 setup_timeMgr (Source File: lisdrv_module.F90)

Initializes the ESMF time manager

INTERFACE:

```
subroutine setup_timeMgr()
```

CONTENTS:

```

integer :: ryr, rmo, rda, rhr !Mainly for GSWP
ryr = 1982
rmo = 7
rda = 1
rhr = 0
dtime = lis%t%ts
ref_ymd=ryr*10000 + rmo*100 + rda
ref_tod=(rhr*3600)
start_ymd=lis%t%syr*10000 + lis%t%smo*100 + lis%t%sda
start_tod=lis%t%shr*3600 + lis%t%smn*60 + lis%t%sss
stop_ymd =lis%t%eyr*10000 + lis%t%emo*100 + lis%t%eda
stop_tod =lis%t%ehr*3600 + lis%t%emn*60 + lis%t%ess
call timemgr_init()
print*, 'time manager initialized..'

```

1.1.3 LIS_ticktime (Source File: lisdrv_module.F90)

Uses the ESMF time manager to handle model timestepping.

INTERFACE:

```
subroutine LIS_ticktime()
```

USES:

```
use driverpardef_module, only: MPI_LT_STRUCT
```

CONTENTS:

```
if(masterproc) then
    call advance_timestep()
    lis%t%tscount = get_nstep()
    call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
    call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
    call updatetime(lis%t) !Updates LDAS variables.
endif
#if (defined SPMD)
    call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
#endif
```

1.1.4 LIS_allocate_memory (Source File: lisdrv_module.F90)

Allocates memory for the domain variables, initializes MPI data structures, and computes domain decomposition

INTERFACE:

```
subroutine LIS_allocate_memory()
```

CONTENTS:

```
if ( masterproc ) then
    nc = getnc()
    nr = getnr()
    maxt = getmaxt()
    call setnch(nc, nr, maxt)
endif
#if ( defined OPENDAP )
    call MPI_BCAST(lis%d%lnc, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%lnr, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%nt, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%mina, 1, MPI_REAL, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%maxt, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
!    call MPI_BCAST(lis%p%koster, 1, MPI_INTEGER, 0, &
!                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%mfile, 50, MPI_CHARACTER, 0, &
```

```

        MPI_COMM_WORLD, ierr)
call MPI_BCAST(lis%p%vfile, 50, MPI_CHARACTER, 0, &
    MPI_COMM_WORLD, ierr)
#endif
#if ( ! defined OPENDAP )
    if(masterproc)  then
#endif
    call domain_init(lis%d%domain)
#endif
#if ( ! defined OPENDAP )
endif
#endif
call def_driverpar_structs()
call MPI_BCAST(lis%d, 1, MPI_LD_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
call MPI_BCAST(lis%f, 1, MPI_LF_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
call MPI_BCAST(lis%p, 1, MPI_LP_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
call MPI_BCAST(lis%o, 1, MPI_LO_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
call MPI_BCAST(lis%a, 1, MPI_LA_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
#endif
#if ( ! defined OPENDAP )
    lis%d%ngrid = lis%d%glbngrid
    lis%d%nch   = lis%d%glbnch
#endif
call allocate_tileddd
if(masterproc) then
    call tile_spmd_init(tile, lis%d%glbnch,lis%f%nmif)
endif
call spread_tdds()
if(.NOT.masterproc) then
    allocate(tile(di_array(iam)))
endif
if(npes>1) then
    call MPI_SCATTERV(tile,di_array,displs, &
        MPI_TILE_STRUCT,tile,di_array(iam),MPI_TILE_STRUCT, &
        0,MPI_COMM_WORLD,ierr)
endif
call allocate_gdd
if(masterproc) then
    call grid_spmd_init(tile,lis%d%glbnch, &
        lis%f%nmif, lis%d%glbngrid)
endif
call spread_gdds()

```

```

if(.NOT.masterproc) then
    allocate(grid(gdi(iam)))
endif
if(npes >1) then
    call MPI_SCATTERV(grid,gdi,gdisp, &
                      MPI_GRID_STRUCT,grid,gdi(iam),MPI_GRID_STRUCT, &
                      0,MPI_COMM_WORLD,ierr)
endif
call dist_gindex()
if(masterproc) then
    call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
    call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
    call updatetime(lis%t)
endif
call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0,  &
               MPI_COMM_WORLD, ierr)

```

1.1.5 setnch (Source File: lisdrv_module.F90)

Computes an estimate of the total number of tiles

INTERFACE:

```
subroutine setnch(nc, nr, maxt)
```

ARGUMENTS:

```

integer::nc
integer::nr
integer::maxt

```

CONTENTS:

```
lis%d%glbnch = nc*nr*maxt !!may be too big
```

1.1.6 getdomain (Source File: lisdrv_module.F90)

Returns the domain resolution

INTERFACE:

```
function getdomain() result(d)
```

ARGUMENTS:

```
integer :: d
```

CONTENTS:

```
d = lis%d%domain
```

1.1.7 getlsm (Source File: lisdrv_module.F90)

Returns which land surface model is executed

INTERFACE:

```
function getlsm() result(lsmno)
```

ARGUMENTS:

```
integer :: lsmno
```

CONTENTS:

```
lsmno = lis%d%lsm
```

1.1.8 getnch (Source File: lisdrv_module.F90)

Returns the number of model tiles

INTERFACE:

```
function getnch() result(n)
```

ARGUMENTS:

```
integer :: n
```

CONTENTS:

```
n = lis%d%glbnch
```

1.1.9 getnc (Source File: lisdrv_module.F90)

Returns the number of columns

INTERFACE:

```
function getnc() result(ncol)
```

ARGUMENTS:

```
integer :: ncol
```

CONTENTS:

```
ncol = lis%d%lnc
```

1.1.10 getnr (Source File: lisdrv_module.F90)

Returns the number of columns

INTERFACE:

```
function getnr() result(nrow)
```

ARGUMENTS:

```
integer :: nrow
```

CONTENTS:

```
nrow = lis%d%lnr
```

1.1.11 getmaxt (Source File: lisdrv_module.F90)

Returns the maximum number of tiles

INTERFACE:

```
function getmaxt() result(maxt)
```

ARGUMENTS:

```
integer :: maxt
```

CONTENTS:

```
maxt = lis%d%maxt
```

1.1.12 getforcing (Source File: lisdrv_module.F90)

Returns the type of forcing used

INTERFACE:

```
function getforcing() result(f)
```

ARGUMENTS:

```
integer:: f
```

CONTENTS:

```
f = lis%f%force
```

1.1.13 getnmif (Source File: lisdrv_module.F90)

Returns the number of forcing variables for model initialization

INTERFACE:

```
function getnmif() result(f)
```

ARGUMENTS:

```
integer:: f
```

CONTENTS:

```
f = lis%f%nmif
```

1.1.14 LIS_endofrun (Source File: lisdrv_module.F90)

Returns if the end of simulation has reached

INTERFACE:

```
function LIS_endofrun() result(finish)
```

ARGUMENTS:

```
logical :: finish
integer :: ierr
```

CONTENTS:

```
if(masterproc) then
    finish = is_last_step()
endif
#if (defined SPMD)
    call MPI_BCAST(finish, 1, MPI_LOGICAL, 0, &
                  MPI_COMM_WORLD, ierr)
#endif
```

1.1.15 dist_gindex (Source File: lisdrv_module.F90)

Distributes the mask indices on compute nodes for a GDS-based execution

INTERFACE:

```
subroutine dist_gindex
```

USES:

```
use grid_spmdMod
```

```
implicit none
```

ARGUMENTS:

```
integer :: findex, lindex, nr_count, t
integer :: ierr, status(MPI_STATUS_SIZE)
integer :: grid_offset, grid_lb, grid_ub
integer :: i, j
```

CONTENTS:

```
#if ( defined OPENDAP )
  if ( npes > 1 ) then
    if ( masterproc ) then
      do t = 1, npes-1
        findex = tile( gdisp(t) + 1 )%row
        lindex = tile( gdisp(t) + gdi(t) )%row
        nr_count = (lindex-findex+1)
        print*, 'DBG: lisdrv_module -- ', &
                  't, findex,lindex,lis%d%gnc,nr_count', &
                  t, findex,lindex,lis%d%gnc,nr_count, ' (, iam, )'
        call mpi_send(nr_count,1,MPI_INTEGER,t,t, &
                     MPI_COMM_WORLD,ierr)
        call mpi_send(glbgindex(:,findex:lindex), &
                     lis%d%gnc*nr_count, &
                     MPI_INTEGER,t,t,MPI_COMM_WORLD,ierr)
      enddo
    else
      call mpi_recv(nr_count,1,MPI_INTEGER,0,MPI_ANY_TAG, &
                   MPI_COMM_WORLD,status,ierr)
      allocate(gindex(lis%d%gnc,nr_count),stat=ierr)
      call check_error(ierr,'lisdrv_module', &
                       'Error allocating gindex.',iam)
      call mpi_recv(gindex,lis%d%gnc*nr_count,MPI_INTEGER,0, &
                   MPI_ANY_TAG, &
                   MPI_COMM_WORLD,status,ierr)
    endif
  endif
```

```

if ( masterproc ) then
    findex = tile( gdisp(0) + 1 )%row
    lindex = tile( gdisp(0) + gdi(0) )%row
    nr_count = (lindex-findex+1)
    print*, 'DBG: lisdrv_module -- ', &
        'findex,lindex,lis%d%gnc,nr_count', &
        'findex,lindex,lis%d%gnc,nr_count,' (', iam, ')
    allocate(gindex(lis%d%gnc,nr_count),stat=ierr)
    call check_error(ierr,'lisdrv_module', &
        'Error allocating gindex.',iam)
    print*, 'DBG: lisdrv_module -- size(gindex),size(glbindex)', &
        'size(gindex(1:lis%d%gnc,1:nr_count)), &
        'size(glbindex(1:lis%d%gnc,finex:lindex))
    gindex = glbindex(:,finex:lindex)
endif

grid_lb = gdisp(iam) + 1
grid_ub = gdisp(iam) + gdi(iam)
grid_offset = gdisp(iam)
do j = 1, nr_count
    do i = 1, lis%d%gnc
        if ( gindex(i,j) /= -1 ) then
            if ( ( gindex(i,j) < grid_lb ) .or. &
                ( gindex(i,j) > grid_ub ) ) then
                gindex(i,j) = - 1
            else
                gindex(i,j) = gindex(i,j) - grid_offset
            endif
        endif
    enddo
enddo
#else
if ( masterproc ) then
    allocate(gindex(lis%d%lnc,lis%d%lnr),stat=ierr)
    call check_error(ierr,'lisdrv_module', &
        'Error allocating gindex.',iam)
    gindex=glbindex
endif
#endif

```